**Q1: (chapter 11)**

**Create a base class called Vehicle that has the manufacturer's name (string), number of cylinders in the engine (int) and owner (type Person class). Then create a class called Truck that is derived from Vehicle and has additional properties: the load capacity in tons (double) and towing capacity in pounds (int). Be sure your classes have a reasonable complement of constructors, accessor and mutator member functions.  Person class has a private member name (string) and the following public methods: no-arg constructor, constructor passing name as a string, getName returning string. Define Car class that is derived from the Vehicle class. Define a class called SportCar that is derived from Car class. Choose the appropriate member variables and methods. Write a driver program that test all your member functions.**

Person Class

**public** **class** Person {

**private** String owner;

Person(){

owner = " ";

}

Person (String newOwner){

owner = newOwner;

}

//Setter

**public** **void** setOwner(String newName) {

owner = newName;

}

// getter

**public** String getOwner() {

**return** owner;

}

}

Vehicle Class

**public** **class** Vehicle {

**private** String manufacture;

**private** **int** noOfCylinders;

**private** Person owner;

Vehicle(){

owner=**new** Person();

manufacture = " ";

noOfCylinders = 0;

}

Vehicle (String newManufacture, **int** newCylinders , String newOwner ){

manufacture = newManufacture;

noOfCylinders = newCylinders;

owner=**new** Person(newOwner);

}

//copy Constructor

Vehicle (Vehicle v1){

**this**.manufacture = v1.getManufacutre();

**this**.noOfCylinders = v1.getCylinders();

owner.setOwner(getVehicleOwner());

}

//setter

**public** **void** setVehicleOwner (String newOwner) {

owner.setOwner(newOwner);

}

**public** **void** setManufacture (String newManufacture) {

manufacture = newManufacture;

}

**public** **void** setCylindes(**int** newCylinders) {

**this**.noOfCylinders = newCylinders;

}

**public** **void** setOwner(String newOwner) {

owner.setOwner(newOwner);

}

//getter

**public** String getManufacutre() {

**return** manufacture;

}

**public** **int** getCylinders() {

**return** noOfCylinders;

}

**public** String getVehicleOwner() {

**return** owner.getOwner();

}

**public** String toString() {

**return** "Manfacture: " + **this**.manufacture+ " "

+"No of Cylinders: " + **this**.noOfCylinders+ " "

+ "Owner: " + **this**.owner.getOwner() ;

}

}

Truck Class

**public** **class** Truck **extends** Vehicle {

**private** **double** loadCapacity ;

**private** **int** towingCapacity;

Truck(){

loadCapacity =0.0;

towingCapacity =0;

**super**.setCylindes(0);

**super**.setManufacture(**null**);

**super**.setVehicleOwner(**null**);

}

Truck(String newManu,**int** newCylinder,**double** newLoad, **int** newTowing,String newOwner ){

loadCapacity = newLoad;

towingCapacity = newTowing;

**super**.setVehicleOwner(newOwner);

**super**.setCylindes(newCylinder);

**super**.setManufacture(newManu);

}

//Copy Constructor

Truck (Truck t1){

**super**.setCylindes(t1.getCylinders());

**super**.setManufacture(t1.getManufacutre());

**super**.setVehicleOwner(t1.getVehicleOwner());

**this**.loadCapacity = t1.getLoad();

**this**.towingCapacity = t1.getTowing();

}

//setter

**public** **void** setLoad(**double** newLoad) {

loadCapacity = newLoad;

}

**public** **void** setTowing (**int** newTowing) {

towingCapacity = newTowing;

}

//getter

**public** **double** getLoad() {

**return** loadCapacity;

}

**public** **int** getTowing() {

**return** towingCapacity;

}

@Override

**public** String toString() {

**return** "Manufacture: "+ **super**.getManufacutre() +" "+

"No of Cylinders: " + **super**.getCylinders() + " "+

"Owner: "+ **super**.getVehicleOwner() + " "+

"Load Capacity: " + **this**.loadCapacity + " " +

"Towing Capacity: " + **this**.towingCapacity ;

}

}

Car Class

**public** **class** Car **extends** Vehicle {

**private** **int** noOfSeats;

**private** **int** noOfDoors;

Car(){

noOfSeats = 0;

noOfDoors = 0;

**super**.setCylindes(0);

**super**.setManufacture(**null**);

**super**.setVehicleOwner(**null**);

}

Car(String newManufacture, **int** newCylinders ,**int** newSeats, **int** newDoors, String newOwner){

noOfDoors = newDoors;

noOfSeats = newSeats;

**super**.setVehicleOwner(newOwner);

**super**.setManufacture(newManufacture);

**super**.setCylindes(newCylinders);

}

//setter

**public** **void** setDoors(**int** doors) {

noOfDoors = doors;

}

**public** **void** setSeats (**int** seats) {

noOfSeats = seats;

}

//getter

**public** **int** getDoors () {

**return** noOfDoors;

}

**public** **int** getSeats() {

**return** noOfSeats;

}

@Override

**public** String toString() {

**return** "Manufacture: " + **super**.getManufacutre() + " "

+ "No of Cylinder: "+ **super**.getCylinders() + " "

+ "Owner: "+ **super**.getVehicleOwner()+" "

+ "No of Doors " + **this**.noOfDoors +" "

+ "No of Seats " + **this**.noOfSeats ;

}

}

SportsCar Class

**public** **class** SportsCar **extends** Car {

**private** **int** horsePower;

SportsCar(){

horsePower = 0;

**super**.setDoors(0);

**super**.setSeats(0);

**super**.setCylindes(0);

**super**.setManufacture(**null**);

**super**.setVehicleOwner(**null**);

}

SportsCar(String newManufacture, **int** newCylinders ,**int** newSeats, **int** newDoors, String newOwner, **int** newHorse){

horsePower = newHorse;

**super**.setVehicleOwner(newOwner);

**super**.setManufacture(newManufacture);

**super**.setCylindes(newCylinders);

**super**.setSeats(newSeats);

**super**.setDoors(newDoors);

}

//setter

**public** **void** setHorsePower (**int** newHP ) {

horsePower = newHP;

}

//getter

**public** **int** getHorsePower () {

**return** horsePower;

}

@Override

**public** String toString() {

**return** "Manufacture: " + **super**.getManufacutre() + " "

+ "No of Cylinder: "+ **super**.getCylinders() + " "

+ "Owner: "+ **super**.getVehicleOwner()+" "

+ "No of Doors " + **super**.getDoors() +" "

+ "No of Seats " + **super**.getSeats() + " "

+ "HorsePower " + **this**.horsePower;

}

}

Driver Class

**public** **class** Driver {

**public** **static** **void** main (String args []) {

//SportsCar(String newManufacture, int newCylinders ,int newSeats, int newDoors, String newOwner, int newHorse)

SportsCar sCar1 = **new** SportsCar("McLaren", 8 , 2 ,2 ,"Naomi", 903 );

System.***out***.println("Sports Cars");

System.***out***.println(sCar1 + "\n");

//Car(String newManufacture, int newCylinders ,int newSeats, int newDoors, String newOwner, )

Car c2 = **new** Car();

c2.setManufacture("Toyota");

c2.setCylindes(4);

c2.setSeats(4);

c2.setDoors(4);

c2.setVehicleOwner("James");

System.***out***.println("Car");

System.***out***.println(c2 + "\n");

//Truck(String newManu,int newCylinder,double newLoad, int newTowing,String newOwner )

Truck t3 = **new** Truck ("Ford", 6 , 1485 , 7700, "Nicolas" );

System.***out***.println("Truck");

System.***out***.println(t3 + "\n");

//Vehicle (String newManufacture, int newCylinders , String newOwner )

Vehicle v4 = **new** Vehicle ("BMW",6, "Boris");

System.***out***.println("Vehicle");

System.***out***.println(v4 + "\n");

// Person

Person p5 = **new** Person();

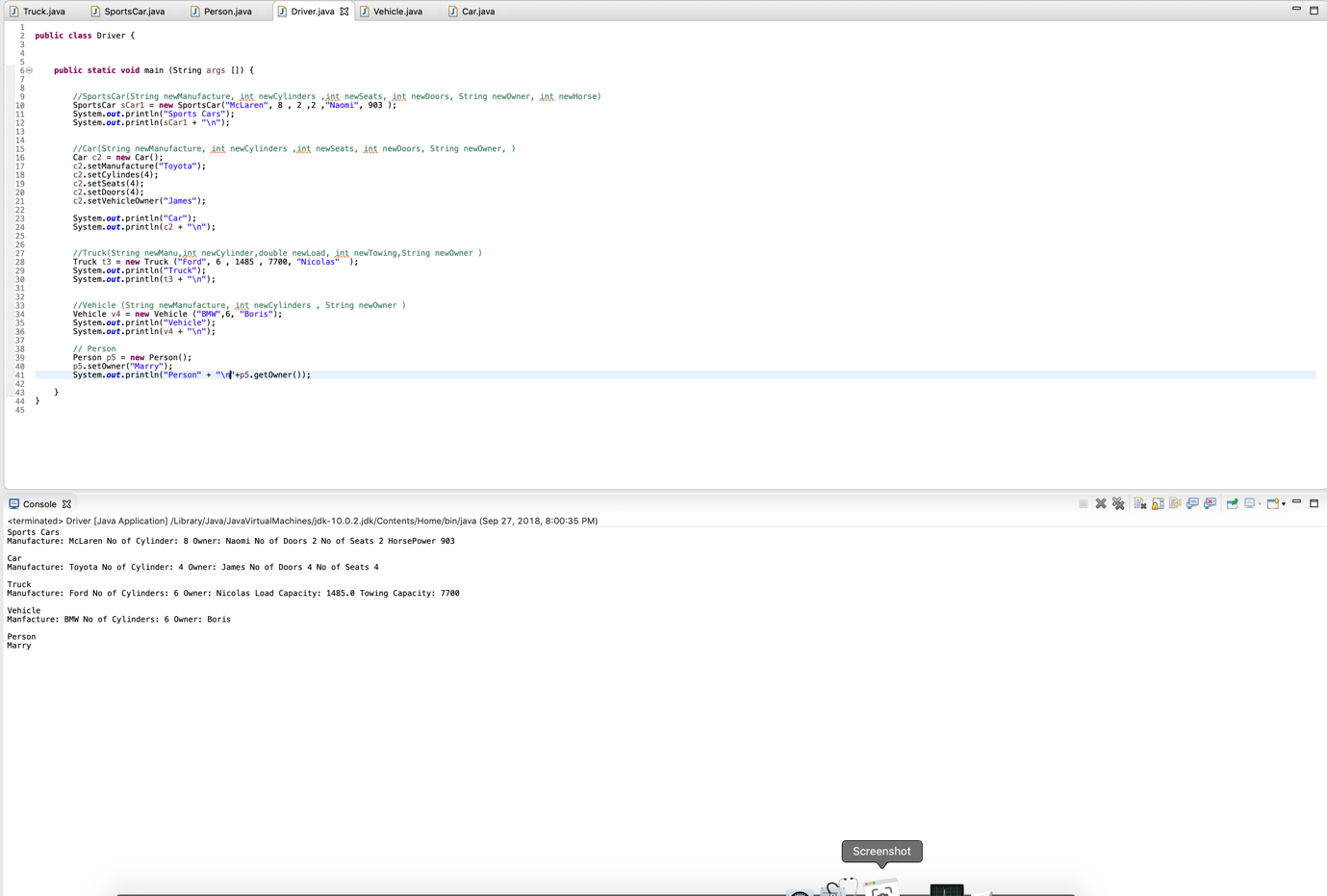
p5.setOwner("Marry");

System.***out***.println("Person" + "\n"+p5.getOwner());

}

}

Screenshot

****

**Q2: (chapter 11)**

**Consider a graphics system that has classes for various figures- rectangles, squares, triangles, circles and son on. For example, a rectangle might have data members for height, width and center points, while a square and circle might have only a center point and an edge length or radius, respectively. In a well-designed system, these would be derived from a common class, Figure. You are to implement such a system.**

**The class Figure is the base class. You should add only Rectangle and Triangle classes derived from Figure. Each class has stubs for member functions erase and draw. Each of these member function outputs a message telling what function has been called and what the class of the calling object is. Since these are just stubs, they do nothing more than output this message. The member function center calls the erase and draw functions to erase and redraw the figure at the center. Since you have only stubs for erase and draw, center will not do any "centering" but will call the member functions erase and draw. Also add an output message in the member function center that announces that center is being called. The member functions should take no arguments. Write a driver program to test the classes.**

Figure Class

**package** Question2;

**public** **class** Figure {

**private** **int** centerPoint;

Figure(){

}

Figure (**int** cent ){

centerPoint = cent;

}

//Setters

**public** **void** setCpoint(**int** c ) {

centerPoint = c ;

}

//Getters

**public** **int** getCpoint() {

**return** centerPoint;

}

//Methods

//Erase

**public** String erase() {

**return** "Figure Class calling Erase";

}

//draw

**public** String draw() {

**return** "Figure Class calling Draw";

}

//center

**public** String center() {

erase();

draw();

**return** "Figure Class calling Center";

}

}

Rectangle Class

**package** Question2;

**public** **class** Rectangle **extends** Figure {

**private** **int** width;

**private** **int** height;

Rectangle (){

}

Rectangle (**int** newWidth, **int** newHeight,**int** newCenter ){

width = newWidth;

height = newHeight;

**super**.setCpoint(newCenter);

}

//setter

**public** **void** setHeight(**int** newH) {

height = newH;

}

**public** **void** setWidth (**int** newW) {

width = newW;

}

//getter

**public** **int** getHeight() {

**return** height;

}

**public** **int** getWidth() {

**return** width;

}

//Methods

//Erase

@Override

**public** String erase() {

**super**.erase();

**return** "Rectangle Class calling Erase";

}

//draw

@Override

**public** String draw() {

**super**.draw();

**return** "Rectangle Class calling Draw";

}

}

Triangle Class

**package** Question2;

**public** **class** Triangle **extends** Figure{

**private** **int** base ;

**private** **int** side ;

Triangle (){

}

Triangle (**int** newBase , **int** newSide , **int** Cent ){

base = newBase;

side = newSide;

**super**.setCpoint(Cent);

}

//setter

**public** **void** setBase (**int** b) {

base = b;

}

**public** **void** setSide (**int** s) {

side = s;

}

//getter

**public** **int** getBase () {

**return** base;

}

**public** **int** getSide () {

**return** side;

}

//Methods

//Erase

@Override

**public** String erase() {

**super**.erase();

**return** "Triangle Class calling Erase";

}

//draw

@Override

**public** String draw() {

**super**.draw();

**return** "Triangle Class calling Draw";

}

}

Driver Class

**package** Question2;

**public** **class** Driver {

**public** **static** **void** main(String[] args) {

Rectangle r1 = **new** Rectangle ();

Triangle t1 = **new** Triangle();

Figure f1 = **new** Figure();

System.***out***.println("Testing all methods for Rectangle:");

System.***out***.println(r1.draw());

System.***out***.println(r1.erase());

System.***out***.println(r1.center());

System.***out***.println("\nTesting all methods for Triangle:");

System.***out***.println(t1.draw());

System.***out***.println(t1.erase());

System.***out***.println(t1.center());

System.***out***.println("\nTesting all methods for Figure:");

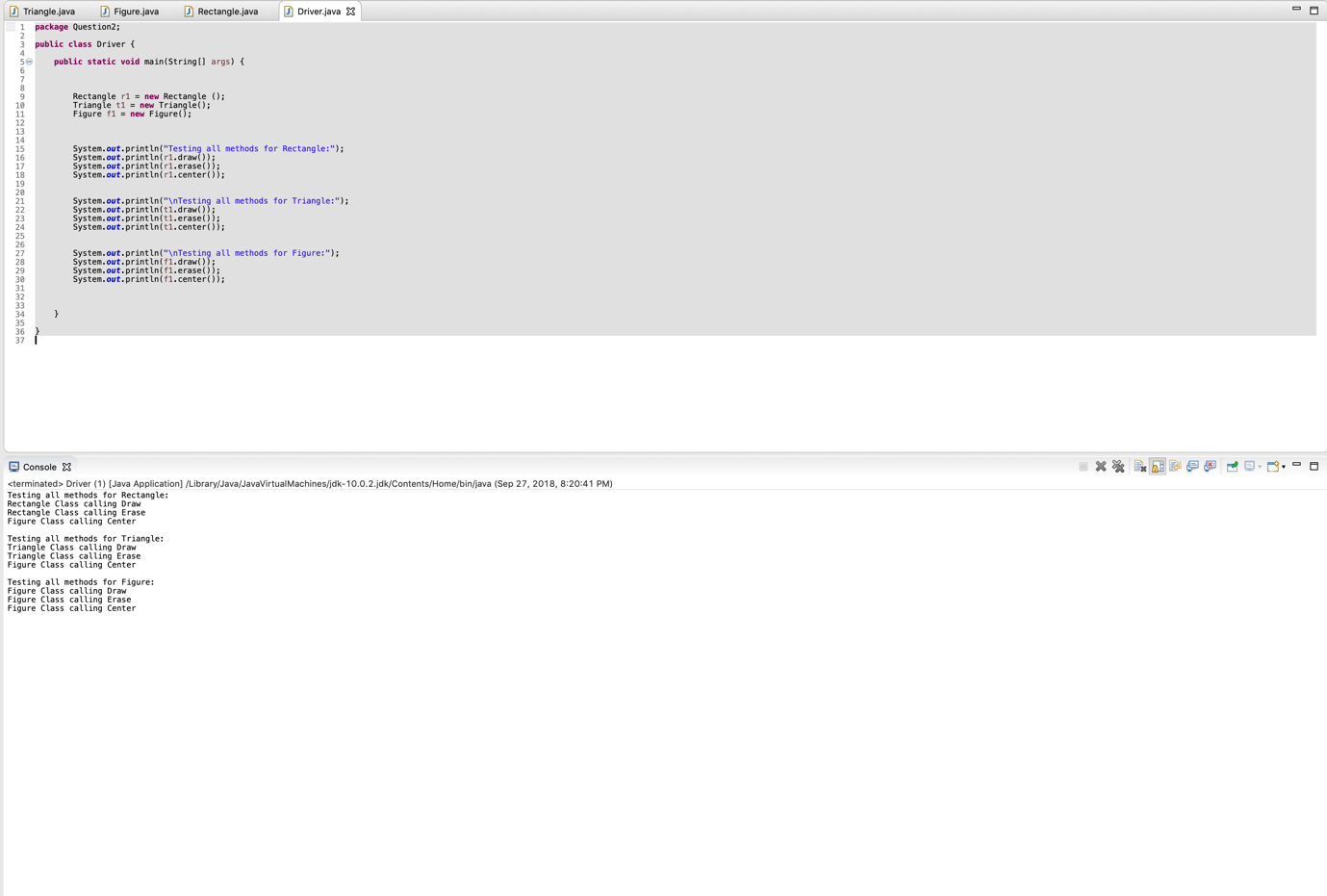
System.***out***.println(f1.draw());

System.***out***.println(f1.erase());

System.***out***.println(f1.center());

}

}



**Q3: (chapter 12)**

**Write a program that converts 24-hour time to 12-hour time. The following is a sample dialogue**

**Enter time in 24-hour notation:**

**13:07**

**That is the same as**

**1:07 PM**

**Again?(Y/N)**

**y**

**Enter time in 24-hour notation:**

**10:65**

**There is no such time as 10:65**

**Try again:**

**Enter time in 24-hour notation:**

**13:07**

**That is the same as**

**4:05 PM**

**Again?(Y/N)**

**n**

**END of PROGRAM**

**You will define an exception class called TimeFormatMistake. If the user enters an illegal time, like 10:65 or even gibberish like 8&\*64, then your program will throw and catch a TimeFormatMistake.**

TimeFormatMistake Class

**package** Question3;

**public** **class** TimeFormatMistake **extends** Exception {

**public** TimeFormatMistake (String message) {

**super** (message);

}

}

ConvertTime Class

**package** Question3;

**import** java.util.Scanner;

**public** **class** ConvertTime {

**public** **static** **void** main (String [] args) {

**boolean** exit = **true**;

**do** {

Scanner input = **new** Scanner (System.***in***);

System.***out***.println("Enter time in 24-hour notation");

String data = input.next();

**try** {

*Converter*(data);

exit = *again*();

} **catch** (TimeFormatMistake e) {

System.***out***.println(e.getMessage());

exit = *again*();

}

}

**while** (exit);

}

**public** **static** **boolean** again() {

Scanner input1 = **new** Scanner (System.***in***);

System.***out***.println("Again?(Y/N)");

String st1 = input1.next();

st1 = st1.toUpperCase();

**if** (st1.charAt(0) == 'Y') {

**return** **true**;

}

**else** {

System.***out***.println("END of Program");

input1.close();

**return** **false**;

}

}

**public** **static** **boolean** validation(String data) {

**boolean** isValid =

(data.length() == 5) &&

(Character.*isDigit*(data.charAt(0))) &&

(Character.*isDigit*(data.charAt(1))) &&

(data.charAt(2) == ':' ) &&

(Character.*isDigit*(data.charAt(3))) &&

(Character.*isDigit*(data.charAt(4)));

**return** isValid;

}

**public** **static** **void** Converter(String data) **throws** TimeFormatMistake {

//char [] data2 = new char [5];

**if** (*validation*(data)) {

String [] strArray = data.split(":");

String strHour = strArray[0];

String strMin = strArray[1];

**int** hour = Integer.*valueOf*(strHour);

**int** min = Integer.*valueOf*(strMin);

**if** (*timeValidate*(hour, min)) {

**switch** (hour) {

**case** 0 : System.***out***.println("That is the same as \n " + 12 + ":" + strMin +" AM"); **break**;

**case** 1 : System.***out***.println("That is the same as \n" + 1 + ":" + strMin +" AM"); **break**;

**case** 2 : System.***out***.println("That is the same as \n" + 2 + ":" + strMin +" AM"); **break**;

**case** 3 : System.***out***.println("That is the same as \n" + 3 + ":" + strMin +" AM"); **break**;

**case** 4 : System.***out***.println("That is the same as \n" + 4 + ":" + strMin +" AM"); **break**;

**case** 5 : System.***out***.println("That is the same as \n" + 5 + ":" + strMin +" AM"); **break**;

**case** 6 : System.***out***.println("That is the same as \n" + 6 + ":" + strMin +" AM"); **break**;

**case** 7 : System.***out***.println("That is the same as \n" + 7 + ":" + strMin +" AM"); **break**;

**case** 8 : System.***out***.println("That is the same as \n" + 8 + ":" + strMin +" AM"); **break**;

**case** 9 : System.***out***.println("That is the same as \n" + 9 + ":" + strMin +" AM"); **break**;

**case** 10 : System.***out***.println("That is the same as \n" + 10 + ":" + strMin +" AM"); **break**;

**case** 11 : System.***out***.println("That is the same as \n" + 11 + ":" + strMin +" AM"); **break**;

**case** 12 : System.***out***.println("That is the same as \n" + 12 + ":" + strMin +" AM"); **break**;

**case** 13 : System.***out***.println("That is the same as \n" + 1 + ":" + strMin +" PM"); **break**;

**case** 14 : System.***out***.println("That is the same as \n" + 2 + ":" + strMin +" PM"); **break**;

**case** 15 : System.***out***.println("That is the same as \n" + 3 + ":" + strMin +" PM"); **break**;

**case** 16 : System.***out***.println("That is the same as \n" + 4 + ":" + strMin +" PM"); **break**;

**case** 17 : System.***out***.println("That is the same as \n" + 5 + ":" + strMin +" PM"); **break**;

**case** 18 : System.***out***.println("That is the same as \n" + 6 + ":" + strMin +" PM"); **break**;

**case** 19 : System.***out***.println("That is the same as \n" + 7 + ":" + strMin +" PM"); **break**;

**case** 20 : System.***out***.println("That is the same as \n" + 8 + ":" + strMin +" PM"); **break**;

**case** 21 : System.***out***.println("That is the same as \n" + 9 + ":" + strMin +" PM"); **break**;

**case** 22 : System.***out***.println("That is the same as \n" + 10 + ":" + strMin +" PM"); **break**;

**case** 23 : System.***out***.println("That is the same as \n" + 11 + ":" + strMin +" PM");**break**;

**case** 24 : System.***out***.println("That is the same as \n" + 12 + ":" + strMin +" AM");**break**;

}

}

**else** {

**throw** **new** TimeFormatMistake ("There is no such time " + data);

}

}

**else** {

**throw** **new** TimeFormatMistake ("There is no such time " + data);

}

}

**public** **static** **boolean** timeValidate(**int** hour, **int** min) {

**if** (hour< 25 && min <60) {

**return** **true**;

}

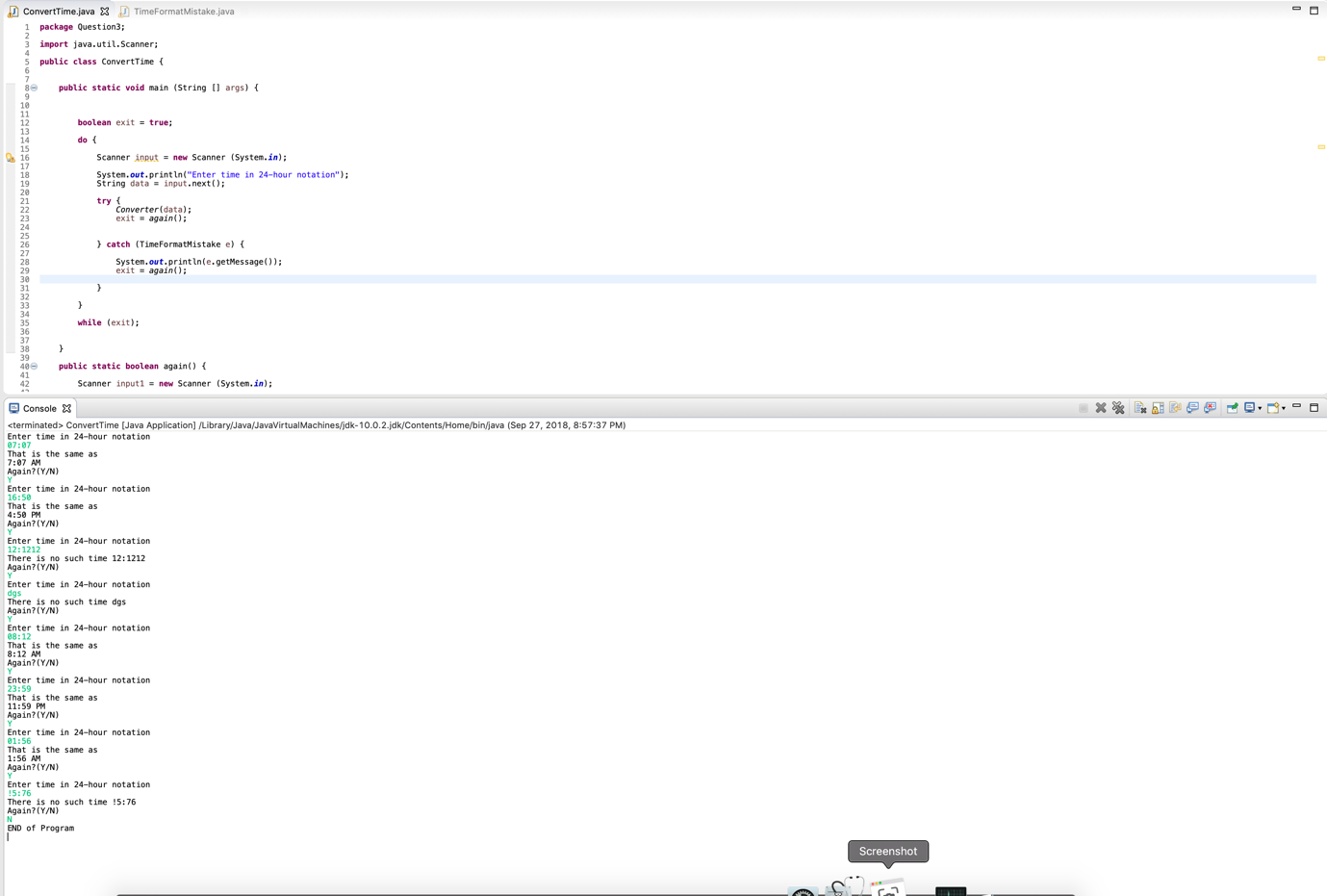
**else** {

**return** **false**;

}

}

}



**Q4: (Chapter 12)**

**Write a program that will search a file of numbers of type double and write the largest and the smallest numbers to the screen. The file contains nothing but numbers separated by blanks or line breaks.**

DoubleReader Class

package Question4;

import java.io.File;

import java.util.InputMismatchException;

import java.util.Scanner;

public class DoubleReader {

public static void main(String [] args) throws Exception {

File file = new File ("/Users/techyouknow/eclipse-workspace/Assignment5/src/Question4/SampleDoubles.txt");

System.out.println("Does the file exists? " + file.exists() );

//counter intialize from 1 ,since captured one value from scanner

int count = 1 ;

double min = 0 ;

double max = 0;

try {

Scanner scan = new Scanner (file);

// int value to start

min = scan.nextDouble();

max = min;

while (scan.hasNextDouble()) {

double test = scan.nextDouble();

count ++;

if (test > max) {

max = test;

}

else if (test < min) {

min = test;

}

}

}

catch (InputMismatchException e) {

System.out.println(e.getMessage());

}

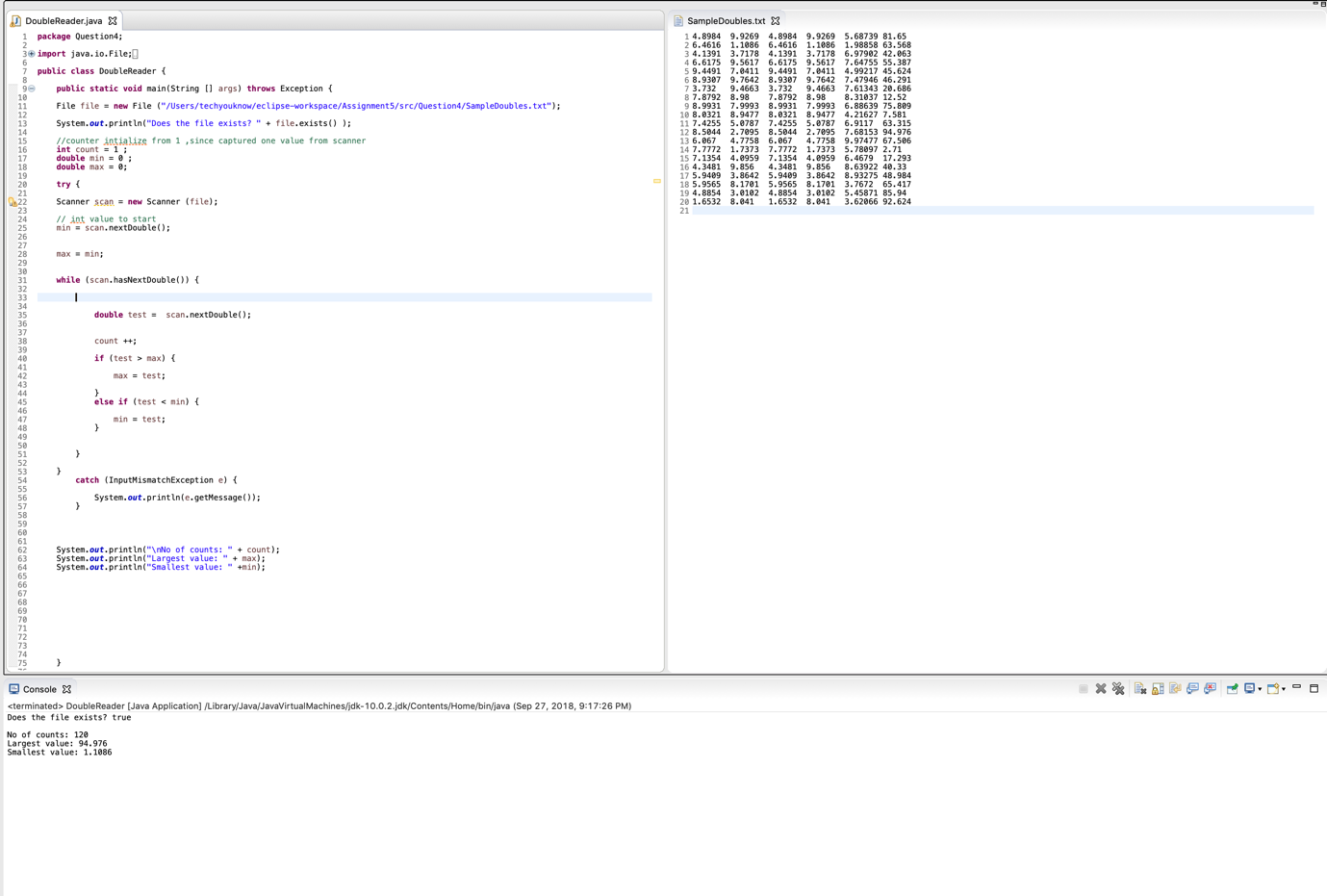
System.out.println("\nNo of counts: " + count);

System.out.println("Largest value: " + max);

System.out.println("Smallest value: " +min);

}

}

****